La primera revista de PC para los desarrolladores de videojuegos



Sumario

Icon Medialab desarrolla la nueva web de Airtel

El nuevo espacio en Internet que acaba de estrenar Airtel se caracteriza por incorporar la tecnología "Lanzadera", hasta ahora única en Internet.
Con un menú de identificación, cuya razón de ser ha sido la sencillez, el usuario en pocos pasos especifica la información en la que está interesado.
En un próximo plazo, Airtel pretende hablar uno a uno con sus clientes y personalizar al máximo su relación gracias a las posibilidades de dicha "Lanzadera". Lo novedoso es que Airtel desea regalar esta tecnología para que pueda reciclarse y adaptarse a cualquier website.
Otras prestaciones del nuevo desarrollo web son: recomendador de terminales, de tarifas, nuevo diseño, navegación de "tercera generación", etc.





La familia Fusión. Vídeo de alta calidad

La segunda generación de la familia Fusion, presentada por la división Digital Infotainment de Rockwell (representada por Diode Electrónica) extiende las características ya ofrecidas con el dispositivo de captura de vídeo Bt848A, el primer chip que se sirve de un mecanismo inteligente DMA y un módulo maestro debus PCI de altas prestaciones para la conexión directa PCI.
Las nuevas capacidades que ofrecen incluyen la captura de audio de TV y de radio FM estéreo, sin necesidad de equipamiento extra o cableado.
También incorpora audición de emisiones de radio estéreo, visualización de programas de TV a elección, envío de video- correo, videoconferencias, recepción de teletexto y servicios VBI, Intercast de Intel y WaveTop de WavePhone. Fusión se encuentra integrada por el chip Bt832 y la cámara digital CMOS QuartSight.



Soporta velocidades de 30 fotogramas por segundo con tamaños CIF (352x288 píxeles) y QCIF (176x144 píxeles). Este dispositivo toma la señal RGB obtenida pr la cámara QuartzSight y realiza un preprocesamiento de la imagen en formato YCrCb compatible con CCIR656 o el popular Vídeo Input Port (VIP).
También ejecuta restauración de color, corrección de gama, conversión de color y, combinado con Fusión, salida de ráfagas de datos CCIR656 hacia el bus maestro PCI de Fusión.
El integrado puede ser utilizado también en otros tipos de subsistemas multimedia (decodificadores de vídeo, controladores de gráficos y dispositivos MPEG II que soporten puerto de entrada CCIR656, por ejemplo).

Fe de errores

En el número anterior de la revista no incluimos dentro del CD el texto que acompañaba al artículo Anarkanoid, de la sección Código Completo. Viene en el CD del presente número.

Tic Tac Toon 2.2

Esta nueva versión ha sido presentada en España por S.G.O, distribuidor del software de dibujos animados Tic Tac Toon.
Esta nueva versión trae una serie de mejoras para mantenerlo entre los mejores puestos del sector de la animación 2D: está basado en vectores y es independiente de la resolución. Estas características han sido destinadas a permitir el trabajo de animación sin papel. Se ha mejorado el trabajo con la base de datos y la gestión de ficheros para que sean más cómodos de usar.
Asimismo, las mejoras en la interfaz de la carta de rodaje permiten renombrar celdas y una visualización más ágil. También se han incluido una gran variedad de atajos de teclado en los módulos de Animation y Paint.
Hay herramientas como *Cortar, Rehacer, Borrar y Aplicar* (modificaciones de color a vectores completos o sólo a una parte de ellos); se ha simplificado el proceso de manejar secuencias de imágenes para facilitar la rotoscopia y la mezcla de secuencias de proyectos; se ha incluido una nueva opción llamada *onion skin* que hace que animar directamente en el sistema y sincronizar la animación multicapa sea más sencillo.

Destacamos

En nuestro CD-Rom de portada incluimos el siguiente material:
- **Los códigos fuentes de los ejemplos comentados en el artículo 3D Manía.**
- **Código completo del juego Wolfenstein 3D, un fabuloso arcade tridimensional.**
- **Informe completo del programa Anarkanoid, aparecido en el número anterior.**

Autor: Jesús de Santos
icg_ent@bigfoot.com

Bump Mapping II: Simulación de Fluidos

Continuamos este mes con técnicas Bump-Mapping. Vamos a dedicar este artículo a la simulación de fluidos y a su visualización gráfica empleando lo que aprendimos en el anterior.

En la simulación de fluidos (a nivel de representación gráfica) nos encontamos con un problema muy similar al que tuvimos cuando empezamos a estudiar el Bump-Mapping. Así es imposible tener un control de una densidad suficiente de partículas para que la visualización sea correcta. Si queremos emplear triángulos para la malla del fluido necesitaremos una resolución excesiva, que generalmente se traducirá en la disminución de resolución poligonal de otros objetos del entorno. No nos podemos permitir este lujo. Para ello vamos a simular el fluido sobre un mapa de alturas, mapa que emplearemos luego al aplicar el Bump-Mapping. Esto explota una de las posibilidades del Bump-Mapping que no mencionamos en el último artículo. El mapa de altura que empleamos para Bump-Mapping puede ser dinámico y actualizarse de *frame* en *frame*.

El principal inconveniente que vamos a encontrar con nuestro método es que realmente es un truco. Es decir, que la ondulación del agua no se produce realmente en 3d. Simplemente simulamos un relieve sobre la textura 2d, por lo que si la ondulación tiene una altura considerable el efecto no quedará bien. Cuando las variaciones de altura se mantienen en un rango pequeño entonces esta técnica funciona a la perfección.

Existen numerosos juegos que emplean esta técnica, no podemos pasar por alto los magníficos efectos de agua que aparecen en *Unreal*. Para un estudio a fondo es necesario "curiosear" un poco por el editor que viene con el juego. La potencia es realmente asombrosa. Por supuesto que todos estos efectos se calculan en tiempo real porque, como veremos un poco más tarde, la simulación de fluidos se puede realizar de un modo muy eficiente, con pocas instrucciones y poco consumo del tiempo de la CPU.

Antes de comenzar a fondo, avisamos que este es quizá el artículo que más desarrollo matemático tiene de los que se han escrito en esta sección. Hemos preferido explicar a fondo el "por qué" del funcionamiento del algoritmo. Existen muchos documentos que hacen referencia a este efecto en Internet, pero muy pocos dan una explicación realmente matemática. Generalmente hay que recurrir a los libros de Física avanzados que requieren un gran conocimiento matemático (si el lector está interesado en el alguno de estos libros, puede consultar por E-mail bibliografía referente a cualquiera de los artículos que en esta sección aparecen). Aun así, no vamos a ser muy estrictos en el desarrollo matemático (y pedimos disculpas a todos los matemáticos que nos lean) porque esta sección es ante todo una sección práctica, como prueba el código que acompaña cada mes a estos artículos. Todo aquel que quiera pasar absolutamente a la práctica y dejarse de aburridos desarrollos puede saltarse los dos siguientes puntos.

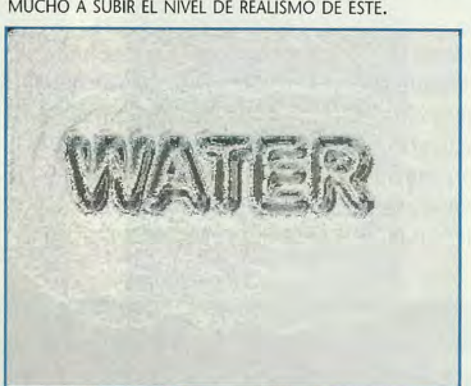


Figura 1. Los resultados que obtenemos mediante la simulacion de fluidos son realmente impresionantes. Su inclusion en un juego ayuda mucho a subir el nivel de realismo de éste.

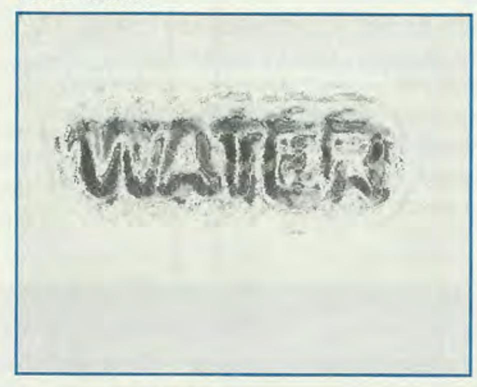


Figura 2. Aparte de las enormes posibilidades de este tipo de simulacion, los efectos que se obtienen son espectaculares y muy agradecidos por el publico.



Figura 3. Uno de los aspectos más atractivos de los simuladores actuales es la imitación del efecto de los líquidos sobre los objetos.

PRELIMINAR

Antes de comenzar con la simulación de fluidos, vamos a recordar algo que nos va a ser de mucha ayuda.

- La definición de derivada primera de una función:

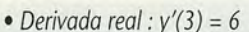
  $df/dh = f' = \lim(h\to 0) ((f(x+h) - f(x)) / h)$ (1)

  Cuando «h» es muy pequeña podemos emplear la siguiente aproximación.

  $df/dh = (f(x+h) - f(x)) / h$ (2)

  Ejemplo: *Sea la función* $y=x^2$. $y'=2x$

  Tomando el punto $x=3$



- *Derivada real : y'(3) = 6*
- *Derivada aproximada h=1 : y'(3) = 4^2 - 3^2 = 7*

- La definición de derivada segunda de una función:

df/dh^2 = f'' = lim(h->0) ((f(x+h) - 2f(x) + f(x-h)) / h^2)
(3)

De nuevo cuando «h» es muy pequeña podemos emplear:

df/dh^2 = (f(x+h) - 2*f(x) + f(x-h)) / h^2
(4)

Ejemplo: *con la función y=x^3. y'=3x^2. y''=6x*

Tomemos otra vez el punto x=3

- *Derivada real : y''(3) = 18*
- *Derivada aproximada : y''(3) = 4^3 - 2*3^3 + 2^3 = 18*

- **Breve noción de ecuación diferencial.** Las ecuaciones diferenciales son de gran utilidad en numerosos campos físicos. Básicamente (perdón a los matemáticos de nuevo) una ecuación diferencial es aquella que tiene entre otros términos «f» y su derivada «f'» (si hablamos de ecuación diferencial de primer orden). Con un ejemplo queda más claro,

*v = dv/dt * t*

En esta ecuación encontramos una dificultad que no aparece en ecuaciones ordinarias, tenemos como términos de la ecuación v, y dv/dt. La solución de una ecuación diferencial es una función.
No vamos a tratar más a fondo estos aspectos porque se salen de los objetivos de esta sección.

SIMULACION DE FLUIDOS

La siguiente ecuación diferencial modeliza bastante bien el comportamiento de fluidos.

du/dt^2 = a(du/dx^2 + du/dy^2)
(5)

Donde u(t,x,y) es una función de tres parámetros: tiempo y posicion bi-dimensional. Vamos a emplear una superficie de dimensiones (x,y) para cada instante de tiempo a simular. Ahora veremos exactamente el número de superficies necesarias, pero de momento basta con comprender que u(t,x,y)



FIGURA 4. LOS MOVIMIENTOS QUE SE REALIZAN DENTRO DE UN FLUÍDO TIENEN UNA "DENSIDAD" DIFERENTE.

significa leer la altura del fluido en la posición <x,y> en el instante de tiempo t.
El factor «a» es una constante que regula la densidad del fluido así como el valor de las fuerzas internas y la atracción gravitatoria entre partículas.
Aplicando (4) sobre (5) en las derivadas de segundo orden que aparecen, tenemos :

du/dt^2 = (u(t+1,x,y) - 2u(t,x,y) + u(t-1,x,y)) =

a (u(t,x+1,y) + u(t,x-1,y) + u(t,x,y+1) + u(t,x,y-1) - 4u(t,x,y))*

De esta ecuación estamos interesados en despejar el término u(t+1) que es el que nos va a indicar las variaciones que se producen a lo largo del tiempo en el fluido a simular.
Despejamos por tanto u(t+1) de la última ecuación:

u(t+1) = a (u(t,x+1,y) + u(t,x-1,y) + u(t,x,y+1) + u(t,x,y-1)) - u(t-1,x,y) + 2u(t,x,y) - 4au(t,x,y)*

Agrupando términos:

u(t+1) = a (u(t,x+1,y) + u(t,x-1,y) + u(t,x,y+1) + u(t,x,y-1)) - u(t-1,x,y) + (2 - 4a) u(t,x,y)*

Si hacemos que el factor (2-4a) se haga 0, podemos simplificar enormemente el algoritmo. (2-4a) es 0, cuando a=1/2.

Haciendo que a=1/2, obtenemos la siguiente fórmula final:

*u(t+1) = (1/2) * (u(t,x+1,y) + u(t,x-1,y) + u(t,x,y+1) + u(t,x,y-1)) - u(t-1,x,y)*
(6)

LISTADO 1

```
for(i=0,cwater=0;i<YRES;i++,cwater+=XRES)
for(j=0;j<XRES;j++) {

    if(j==0) l=*(old_water+cwater+j);
    else l=*(old_water+cwater+j-1);

    if(j==XRES-1) r=*(old_water+cwater+j);
    else r=*(old_water+i*XRES+j+1);

    if(i==0) u=*(old_water+cwater+j);
    else u=*(old_water+cwater-XRES+j);

    if(i==YRES-1) d=*(old_water+i*XRES+j);
    else d=*(old_water+cwater+XRES+j);

    s = (( l + r + u + d) >> 1 ) -
*(new_water+cwater+j);

    s -= (s>>dampening);

    *(new_water+cwater+j) = s;

    }
```

Que en palabras significa lo siguiente: normalizar cada punto con su vecindad (4 vecinos, sin considerar las diagonales) pero en vez de dividir por 4, dividimos por 2 y luego le restamos la posición que tenía en el instante t-1 [estamos calculando el instante t+1].
La fórmula que hemos obtenido es la "famosa" fórmula que solemos encontrar en numerosos documentos sobre fluidos aplicados a la informática gráfica.
Puede parecer después de este resultado que vamos a necesitar 3 superficies para los intervalos de tiempo t-1,t,t+1. Pero si observamos detenidamente la ecuación (6) veremos que el valor de u(t-1,x,y) sólo afecta a u(t+1,x,y) por lo tanto sólo necesitamos 2 superficies. Una que guarda simultáneamente t+1 y t-1, y la que guarda t.
Aplicando estos resultados todavía no obtenemos un acabado final. Es necesario introducir un factor de atenuación de las ondas cuando se transmiten por el medio. Podemos conseguir esto, restando 1/(2^n) del resultado anterior al propio resultado. Un valor bueno para «n» es 4. Con valores superiores conseguimos que la onda tarde más en desaparecer con lo que tenemos un fluido mucho más viscoso. El efecto contrario lo conseguimos con valores de «n» inferiores a 4.
Con todo lo que hemos dicho aquí, estamos en condiciones de implementar nuestra simulación de fluido.

IMPLEMENTACION

Llegamos a la parte interesante. Como en cada artículo, incluimos en el Cd-Rom que acompaña a la revista una aplicación práctica de este artículo. Pasamos a comentar las partes más relevantes del código.
En el anterior artículo empleamos una luz puntual que se podía mover con las teclas del cursor. En este programa (con el fin de mostrar al lector otra forma de usar el Bump-Mapping) empleamos el siguiente código para simular el relieve:

```
*(wvid+cvid+j)=*(wlight+
(((dx+127+(dy+127)*LIGHT_XSIZE))&65535));
```

Que equivale a situar el foco de luz a una distancia infinita (Luz vectorial).
La suma de 127 es para centrar el foco en la posición de la textura que más iluminación tiene. Destacar que estamos usando un mapa de luz de 256x256, mientras que en nuestro anterior ejemplo empleábamos un mapa de 512x512. El mapa de luz que usamos aparece en la figura 6.

LISTADO 2

```
case 2:        // 15 , 16 bpp

    for(i=0,cvid=0;i<YRES;i++,cvid+=XRES)

            for(j=0;j<XRES;j++) {

                    if(cvid+j==0)
left=cvid+j;
            else left=cvid+j-1;

            if(cvid+j==XRES*YRES-1)
right=cvid+j;
            else right=cvid+j+1;

            if(cvid+j<XRES) up=cvid+j;
            else up=cvid+j-XRES;

            if(cvid+j>=XRES*YRES-XRES)
down=cvid+j;
        else down=cvid+j+XRES;

            dx=*(new_water+left) -
*(new_water+right);
        dy=*(new_water+up) -
*(new_water+down);

            *(wvid+cvid+j)=*(wlight+
(((dx+127+(dy+127)*LIGHT_XSIZE))&65535));

        }

    break;
```

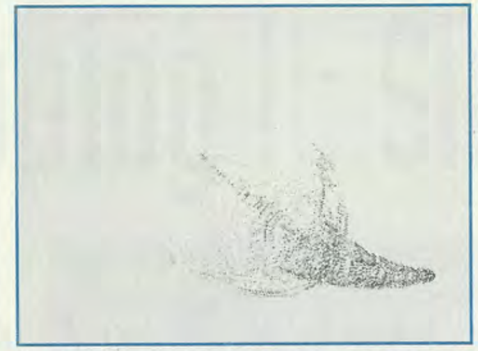


Figura 5. Un objeto que rompe la superficie de un liquido crea una serie de movimientos encadenados en dicho fluído.

La parte importante del programa es la que aparece en el listado número 1. En él se implementa el algoritmo que hemos demostrado en la primera parte del artículo.
El código prácticamente se explica por sí solo. Falta resaltar que no está optimizado en velocidad. Si realmente estamos interesados en la velocidad de esta parte deberíamos:

- Eliminar el chequeo de los casos particulares en el bucle principal. Deben ser tratados fuera del bucle principal.
- Pasar a ensamblador el bucle. Es la única manera de conseguir exprimir los ciclos del procesador. Este algoritmo posee ciertas características que lo hacen fácil de implementar con instrucciones en paralelo en un Pentium.
- Otra diferencia es que el mapa de alturas emplea números con signo. Esto es necesario porque la onda debe oscilar arriba y abajo del nivel base que es el 0. Con un rango de *BYTE* para cada punto de la superficie no hemos obtenido la precisión suficiente por lo que hemos empleado *WORD* para guardar la altura relativa de cada punto de la superficie.
- Como ya hemos comentado, necesitamos 2 superficies para guardar toda la información temporal necesaria. Mostramos aquí la parte de código que carga las superficies y el bloque encargado de intercambiar las superficies temporales.

```
  (u(x,y))
  ...
// Bloques de memoria para las dos superficies

  water1=(TWater
*)malloc(XRES*YRES*sizeof(TWater));
  if(water1==NULL) {
     notify("No hay memoria");
     return FALSE;
  }
  water2=(TWater
*)malloc(XRES*YRES*sizeof(TWater));
  if(water2==NULL) {
```

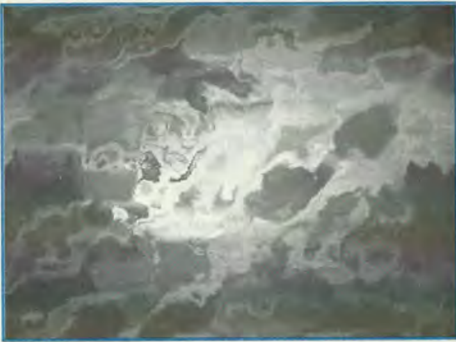


Figura 6. Mapa de luz. Hemos tomado prestada esta textura del editor de Unreal y la hemos modificado segun nuestras necesidades.

```
     notify("No hay memoria");
     return FALSE;
  }
  ...

// Intercambio de Superficies. Una vez por frame,
despues de
// actualizar
  ...
        temp=water1;
        water1=water2;
        water2=temp;
  ...
```

De nuevo hemos incluido la posibilidad de ver el efecto en cualquier resolución de color. En el listado 2 aparece el bucle principal para una resolución de 16 bpp.
No hemos podido evitar incluir un pequeño efecto que apareció por primera vez en la demo HearthQuake de Iguana. Cuando pulsamos espacio, un texto se introduce en el agua poco a poco consiguiendo un efecto super realista. El mapa de alturas que hemos empleado nosotros aparece en la figura 7.
También hemos incluido la posibilidad de afectar al fluido con el ratón. Así, moviendo el cursor lentamente por la ventana, podemos experimentar con nuestro simulador así como con los distintos valores de viscosidad (que podemos variar interactivamente con las teclas +/-).
Hemos empleado el siguiente código para simular el hundimiento progresivo de una zona del mapa de alturas:

```
for(i=0;i<XRES*YRES;i++) {

    p1=(TWater)-(*(bump+i)*5);
    p2=*(water2+i);

    p1=(31*p2+p1)/32;

    *(water2+i)=p1;

}
```

Autor: Antonio Marchal

Resolución de tipos de juegos

Continuamos como cada mes hablando de un tipo de juego. Este mes toca a los del tipo puzzle o Tetris. Este tipo de juegos ha estado en la historia de la humanidad desde tiempos inmemoriables. Por eso no es raro que se hallan hecho versiones de ellos en ordenador, o incluso que se hallan inventado nuevos juegos del tipo puzzle dentro del mundo informatico.

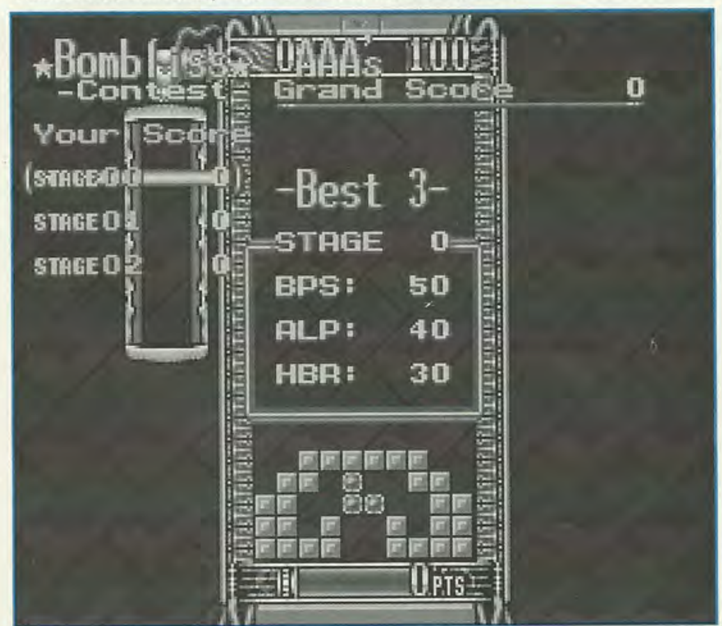
Todo el mundo conoce los puzzles tradicionales. En estos juegos, la inteligencia, retentiva y perspectiva espacial son factores importantes. Pero existen otro tipo de juegos que tienen cierto parecido con éstos. Este es el caso del Tamgran, el cubo de Rubik, el juego de las Parejas, etc. De todos ellos se han hecho versiones para ordenador durante toda la historia de la informática.
Además, desde que aparecieron los juegos de ordenador, ha habido títulos que también cumplían las condiciones de este tipo de juegos, que hemos denominado puzzles.
El caso más conocido de este tipo de juegos es el Tetris. Es tan conocido que incluso se ha estudiado el incorporar la palabra "Tetris" al Diccionario de la Real Academia Española. Las versiones y readaptaciones de la idea original de Alex Pajinov han sido innumerables. Y el ver caer piezas, sean de la forma que sean, en cualquier juego, está a la orden del día.

Pero una vez visto los juegos que se podrían englobar en este grupo, pasemos al estudio de su adaptación al ordenador. Como cada mes, separaremos el artículo en tres secciones, para así facilitar el estudio del mismo. Pero metámonos en faena, dejémonos de preámbulos y que empiecen a caer las piezas.

JUEGOS DEL TIPO PUZZLE Y TETRIS. FUNDAMENTOS

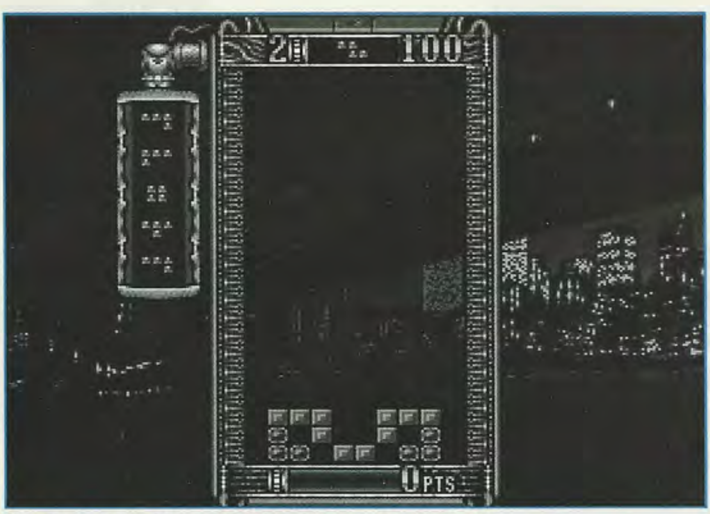
Lo primero, antes de ver los problemas en sí de programación, vamos a ver los denominadores comunes en este tipo de juegos. En todos ellos, se podría denominar misión al objetivo que debemos cumplir. Esta misión se puede hacer de distintas formas, pero suele consistir en juntar, buscar o mover piezas. Por lo tanto, en todos los juegos de puzzles las piezas son una parte importantísima del mismo.
La relación de estas piezas, siguiendo unas normas, daría al juego el ingrediente de inteligencia, que se requiere por el usuario. Pero en otros muchos juegos, para dificultar la misión también se pone un contador de tiempo. Con esto se valorará la rapidez de tomar decisiones que tiene el jugador.
Aparte, se puede hacer un juego de puzzles donde la rapidez de reflejos sea lo que más se valore. Para incluirlos dentro de este tipo de juegos, por supuesto, deben tener un componente de puzzle dentro del mismo, es decir, deben tener piezas.
Y hablando de piezas, estas pueden ser de distinto tipo, tamaño y forma. Debe haber un factor relacional entre las mismas. Es decir, se deben coordinar, por ejemplo, comprobando su color, su tamaño o su posición en el espacio. Pueden ser piezas móviles o estáticas. Y su tamaño puede ser constante o variable. Además pueden también tener distintas formas.
Otro tipo de puzzles, como hemos comentado anteriormente, se integran dentro de otros juegos. El mas usado, sobre todo dentro de videoaventuras, es el que denominamos como conjunto llave- puerta. En estos casos, los puzzles se basan en buscar la llave que abra la puerta correspondiente. Se puede crear un juego, cuya misión consista en buscar sucesivas llaves, que abran distintas puertas, habiendo otras piezas, además de estas. Este podría ser un juego de puzzle puro, pero usando este conjunto de llaves- puerta.

VERSIONES DE VERSIONES, PERO SIEMPRE HAY IDEAS ORIGINALES.

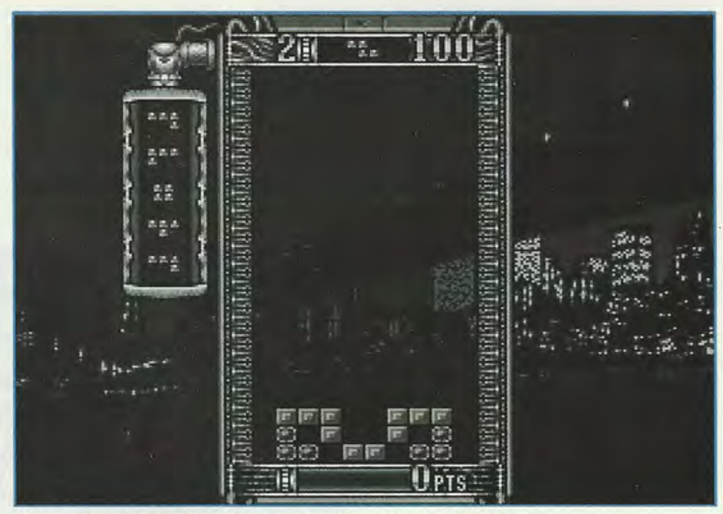


Los puzzles veloces, un género adictivo donde lo que más cuenta son los buenos reflejos

Otro conjunto de piezas muy usado son las parejas, tríos, cuartetos, etc. Es decir, agrupar las piezas, tomando un parámetro, como puede ser su color, forma, o cualquier otro. Para dificultar la misión, se puede poner un tiempo y unas normas de agrupamiento que dificultan la misión. Ejemplos de estos juegos son el Shangai, o el Columns.
Otro juego mítico, de este tipo, es el de Las Parejas, que seria la base de todos ellos.
Además, se puede tener una colocación espacial. Este es el caso del mítico Tetris, que toma líneas, como colocación de piezas. Además, se podrían añadir otros elementos como bombas, bonus, puntos calientes, etc. Estas opciones, han sido tomadas por muchos juegos que son versiones del Tetris pero con mejoras. Se podría reseñar también, dentro de las versiones, el uso de otro



Las piezas, una parte fundamental de cualquier puzzle.



Piezas, que caen, piezas que suben, piezas que se mueven...

tipo de piezas, cambiándole su forma o tomando otro parámetro, como la colocación de las piezas. Por último decir que se podría dar otro ejemplo de colocación espacial que encontraríamos en los típicos puzzles de mesa. De éstos, también se han hecho innumerables versiones.
Existen otro tipo de puzzles, cuyo mayor ejemplo es el Sokoban, donde las piezas forman parte del decorado. Éstas se deben mover, para poder pasar o, como en el juego antes nombrado, colocarlas en una posición correcta. Se han hecho algunas versiones, de este tipo de juego, añadiendo, además de las típicas piezas, otras como bombas, puertas- llave, etc. Incluso, en alguno de ellos, se han añadido enemigos, dándole un aspecto casi de Arcade.
Antes de pasar a ver los problemas que pueden dar el programar estos juegos, constatar un hecho. Algunos de ellos, cuando se cumple la misión, dan un premio, bien en forma de puntos o de forma visual. En este ultimo caso, normalmente se nos distrae con una imagen agradable a la vista. Pero una vez visto todos los denominadores comunes a los juegos de puzzles, veamos la problemática de los mismos.

Los juegos y sus géneros

Se ha abierto una nueva etapa dentro de estos articulos con los que os entretenemos cada mes. El denominador común de todos ellos es el de que tratan de videojuegos, comentando sus fundamentos, viendo su problemática y dando una solución a dichos problemas. Para tener todo el tema más organizado, se ha hecho una agrupación en tipos de videjuegos. La lista completa de esta serie de articulos, que como hemos dicho está organizada por tipos, es la que viene a continuación, aunque siempre puede que haya cambios de última hora, ahí va dicha lista :

- **Arcade de plataformas.**
- **Shot'em'up.**
- **Puzzles y tetris.**
- **Comecocos y juegos especiales.**
- **Juegos de cartas y de mesa.**
- **Aventuras conversacionales.**
- **Rol / RPG.**
- **Juegos de lucha.**
- **Simuladores deportivos.**
- **Simuladores de vuelo.**
- **Simuladores en general.**
- **Estrategia.**
- **Juegos tipo DOOM**

PROBLEMATICA

Como se ha comentado en el anterior apartado, las piezas son una parte esencial en este tipo de juegos. Es por esto, que se hará un estudio de las mismas, para así facilitar la programación. Nos podemos encontrar muchos problemas, algunos de ellos dependiendo del juego por el que hemos optado. Así que veamos cada uno de los aspectos que se pueden incluir dentro de una pieza más detenidamente.
Lo primero es su colocación espacial, o dicho con otras palabras, la posición que ocupa dentro del juego. Se deben tener datos de la posición de la pieza en todo momento. Existen varios métodos, que ya veremos, para poder computar estos datos. Aunque hay que decir que con DIV Games Studios, esto puede resultar muy fácil, siempre y cuando se cumplan unas condiciones.
Dentro de este campo, hay que tener en cuenta si las piezas van a ser movibles, o si por el contrario serán estáticas. Se puede dar el caso de tener dentro del mismo juego los dos tipos de juegos. Es decir, que haya una serie de piezas que se muevan y otras que sean estáticas. O incluso, si se quiere rizar el rizo, se podría dar un tipo de pieza que según qué condiciones sea móvil o no.
En todo momento se debe tener una computación de las mismas. Es decir, un control, para saber con qué pieza estamos tratando en cada momento. Esto se puede hacer de distintas formas, como veremos más adelante. Lo que tiene que quedar claro, es que se debe tener identificada a todas las piezas que componen el puzzle. Sabiendo de qué tipo es y su posición en el espacio, si este es un dato importante.
Otro factor a tener en cuenta, si es que se ha optado por esta modalidad en el juego, es el tiempo. DIV Games Studios, facilita bastante este aspecto, ya que dispone de varios relojes. Aun así, se pueden dar otro tipo de problemas como la cuenta atrás, o de otra índole como puede ser el de controlar los tiempos máximos. Pero todos estos problemas los veremos resueltos en la siguiente sección.
Otro de los problemas que nos podemos encontrar son las fases. Sobre todo si después de cada una de ellas se da un premio al jugador, que normalmente se compone de una imagen o animación. En cualquier caso, se debe tener un control de estos premios, que generará un código. Estudiaremos la forma de hacerlo, utilizando el sistema más sencillo.
Este último problema se puede llevar a un estado máximo. Algo que se ha hecho en algunos juegos y que consiste en tener una especie de base de datos de puzzles. Es decir, que se puedan crear, cargar y guardar puzzles, pudiendo tener todos los puzzles

almacenados en el disco duro. Esta opción trae una gran problemática, que dificulta bastante llevarla a la practica, como luego veremos.
Se podría decir que están vistos todos los problemas que se plantean a la hora de realizar un juego de puzzles. Veamos su resolución punto por punto en el siguiente apartado. Sin más preámbulos, pasemos a dar soluciones a los "puzzles" de programación.

RESOLUCION

Debido a que las piezas son la parte más importante dentro de los puzzles, haremos de cada una de ellas un proceso. Con esto conseguiremos tener unos datos ligados, que facilitarán la programación del videojuego. Se podrá tener un proceso que maneje la totalidad de las piezas o construir el código de proceso para cada tipo en particular.
En cualquiera de los dos casos, algo que se gana al tener las piezas asignadas a procesos, es que su colocación espacial se automatiza de una manera considerable. Esto es así, porque se dispone de las variables locales que poseen todos los procesos y que designan sus coordenadas en pantalla. Puede resultar útil, sobre todo para piezas móviles, ya que alterando sus coordenadas «x» e «y» recolocaremos la pieza.
Pero además se deben tener otros controles sobre las piezas. Es decir, llevar un control. Este se puede hacer de diferentes maneras, utilizando distintos métodos. El primero de ellos es crear una variable local que identifique el tipo de pieza que estamos usando. Esto es muy útil para piezas irregulares, donde la separación del tablero de juego en piezas no está muy definido. Pero si se da el caso,de que utilizamos un tablero de juego que se pueda cuadratizar, se pueden usar otros métodos. En este caso, se entiende cuadratizar como al hecho de que el tablero tenga definidas sus posiciones. Es decir, por ejemplo, en el ajedrez, el tablero de juego seria de 8 por 8 casillas, pudiendo hacer referencia a cualquier casilla por sus coordenadas. En estos casos, es mejor utilizar tablas donde almacenaremos los tipos de piezas. En DIV, sólo se puede utilizar un tipo de tabla, de una dimensión. Esto es solucionable con una simple fórmula del tipo:

*Posicion_en_tabla=x+(y*ancho);*

Y en el caso de utilizar 3 dimensiones, la formula sería:

*Posicion_en_tabla=x+(y*ancho)+(z *ancho*alto);*

Con estas fórmulas se podría tener un acceso a la tabla, como si ésta fuera de más de una dimensión. Mediante valores identificaríamos a cada pieza y en el caso de querer más datos se podría crear una estructura que guardase todos los campos que estuvieran relacionados con la pieza.
Otro aspecto a tener en cuenta en este tipo de juegos, como se ha comentado, es el tiempo. Con DIV, esto es fácil, ya que se dispone de la tabla de variables

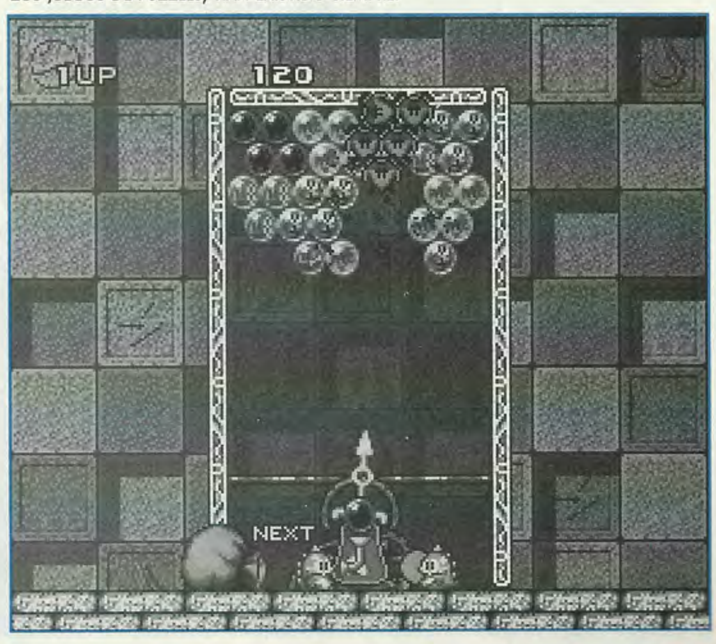
Algo de historia de los juegos de puzzles

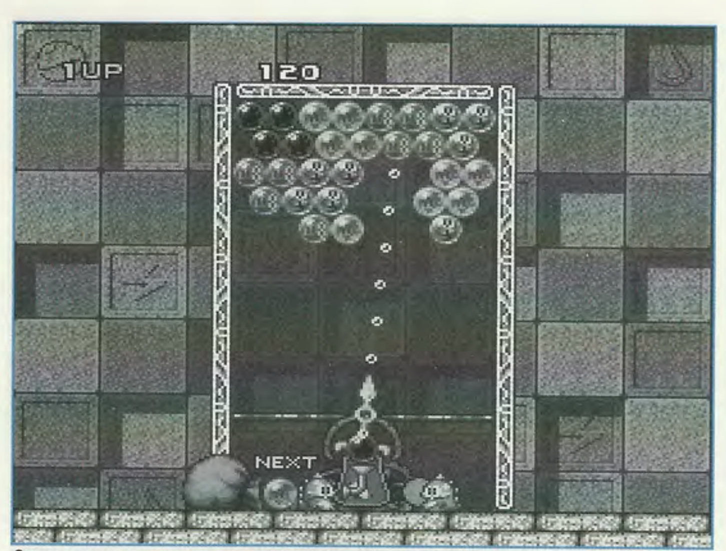
Los juegos, en general, han estado siempre en la historia del ser humano. Y en particular, los juegos de puzzles han tomado una parte importante del tiempo que el hombre ha pasado jugando. Desde los típicos juegos de piezas, cuyo nombre hemos escogido para denominar este tipo de juego, es decir, puzzles, hasta los nuevos retos de inteligencia y retentiva, ha llovido mucho.
Juegos como el Tangram, el cubo de Rubik o los puzzles convencionales, pueden y han sido versionados en ordenador. El número de títulos de videojuegos que son conversiones de estos juegos de mesa es innumerable. Pero el juego tipo puzzle de ordenador por excelencia es el Tetris. Este juego rompió moldes en su día, y todavía sigue en la brecha, sacándose versiones del mismo.
Y hablando de versiones del Tetris, las ha tenido en cantidades industriales. Títulos como Columns, Bomblis, Klax, o Colors han seguido en cierto modo la senda abierta por el de Alex Pajinov. Y es que dentro de los juegos del tipo Puzzle, el hacer versiones con ligeros cambios de otros juegos del mismo tipo, está a la orden del día. Se pueden ver innumerables títulos que bien son idénticos a otros productos anteriores, pero cambiando los gráficos, o bien son ideas remodeladas, pero que guardan la esencia original del puzzle base. Títulos como Puzzle-Bubble, Loopz, o los anteriormente nombrados, se podrían integrar dentro de los juegos que son ideas de otras ideas.
También se puede dar el caso que algunos juegos,que en principio se fundamentan en Arcades o del tipo plataformas, o cualquier otro, pero en algunas de sus fases integren puzzles, para conseguir así una trama más entretenida y variada. Pero esto se puede hacer con todos los tipos, dejando esta elección al diseñador del juego. Reseñar, como mezcla curiosa, la que han elegido algunas casas desarrolladoras, que consiste en mezclar imágenes con alto contenido erótico, o incluso pornográfico, con juegos del tipo puzzle. En estos casos se suele premiar al jugador con una imagen cada vez que acaba un puzzle.

HAN HECHO NUMEROSAS VERSIONES DEL TETRIS.

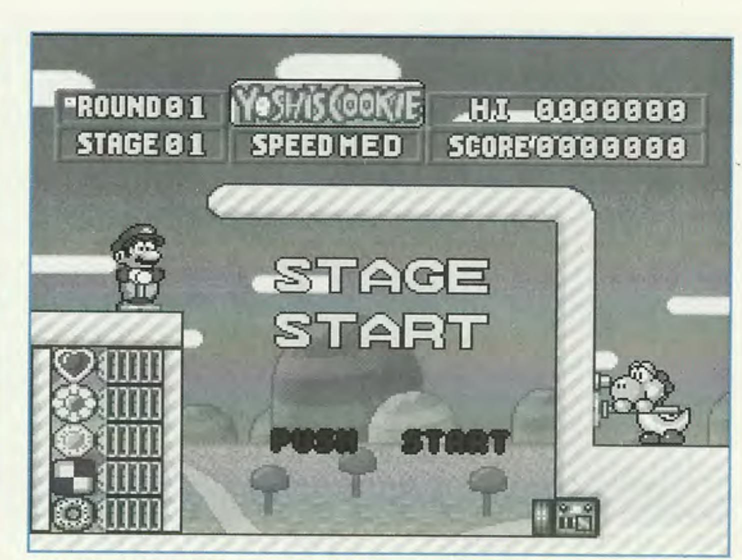


LOS JUEGOS DE PUZZLES, HAN EXISTIDO SIEMPRE.





SIEMPRE PODEMOS PONER RETOS A NUESTRA INTELIGENCIA.



LAS IDEAS SOBRE PUZZLES HAN IDO EVOLUCIONANDO.

globales *timer*[]. Éstas son, en verdad, una serie de 10 relojes, en centésimas de segundos, que nos permite tener todo tipo de cronómetros. Cuando se haga uso de éstas funciones, se debe tener en cuenta que nunca se debe comprobar el tiempo justo, sino hacer uso del símbolo mayor. Es decir, si se quiere comprobar si se ha llegado a los cinco segundos, en vez de hacerlo de la siguiente manera:

```
IF (timer[0]==500)
END
```

Sería más conveniente,utilizar el siguiente método:

```
IF (timer[0]=>500)
END
```

Esto es así, porque es muy difícil pillar a los contadores en un punto determinado, puesto que los contadores se actualizan constantemente. Al ir en centésimas de segundo, es muy difícil atrapar al contador en una centésima de segundo determinada.
Otro aspecto a tener en cuenta, en cronómetros, es la cuenta atrás. No existen herramientas que se hayan creado para este menester. Aun así, usando variables secundarias, se pueden conseguir estos efectos.
Una cuestión destacable son las fases, sobre todo si en éstas se dan premios, en forma visual. Si estos son pantallas, se pueden guardar todas ellas dentro un mismo fichero FPG. Tiene la ventaja de que se son cambiables únicamente modificando el fichero donde se guardan. También se pueden tener colocados dentro del disco duro y usar tablas para hacer referencias a los mismos. Si se quiere crear una especie de base de datos de puzzles, se puede hacer usando este último método. Se recomienda esta operación, ya que el uso de cambio de directorios y de carga de ficheros sin conocer su nombre, es bastante dificultoso desde DIV. Sólo quedaría como ultimo problema el comprobrar si se ha ganado o no. Esto, si se hace uso de tablas, es fácil, ya que únicamente se deben comprobar todas las casillas, mirándolas una a una. Pero en el caso de hacer uso de procesos, lo mejor es tener una variable global que haga de contador de piezas. Cuando se cree una pieza, el contador se incrementará y cuando se elimine, se descontará. También se puede hacer esta utilización, ya que dispone de un contador actualizado.

Resumiendo

Este mes hemos visto otro tipo de juego: el género de los puzzles. El mes que viene tendremos más. Espero vuestras dudas y sugerencias, como todos los meses, en el email: tizo@100mbps.es. Bueno, nada más, espero que os hayan ayudado estas pistas a realizar vuestro juego. Y que os DIVirtáis programando.

Autor: José Antonio Guerra
balder mindless.com

Clippers y Paletas

Este mes vamos a aprender a manejarnos con dos de los componentes más usados de DirectDraw; practicaremos con las paletas de color de los modos indexados y usaremos ventanas de recorte para crear interesantes efectos.

Las *surfaces* paletizadas necesitan una paleta para ser visualizadas correctamente. Una *surface* paletizada es una simple colección de números en la cual cada uno de dichos números corresponde a un pixel. El valor del número es un índice a una tabla de color que indica a DirectDraw qué color usar al visualizar el pixel.

Los objetos *DirectDrawPalette* ofrecen una sencilla manera de gestionar paletas. Un objeto *DirectDrawPalette* representa una tabla de color con 2, 4, 16 o 256 entradas (1, 2, 4 y 8 bits, respectivamente). Cada entrada de la tabla representa un color RGB de 16 o 24 bits, y las paletas de 16 colores pueden asimismo contener referencias a entradas de otra tabla de 8 bits de profundidad. Las paletas pueden ser utilizadas en texturas, *surfaces* no visibles y, por supuesto, en la *surface* principal.

Las paletas pueden ser creadas invocando a *IDirectDraw2::CreatePalette*. Dicha llamada devuelve un puntero a un interfaz *IDirectDrawPalette*, cuyos métodos usaremos para gestionar la tabla de color.

Una paleta se aplica a una *surface* usando el método *IDirectDrawsurface3::SetPalette*. Las paletas, al ser objetos independientes, pueden ser usadas en varias *surfaces*.

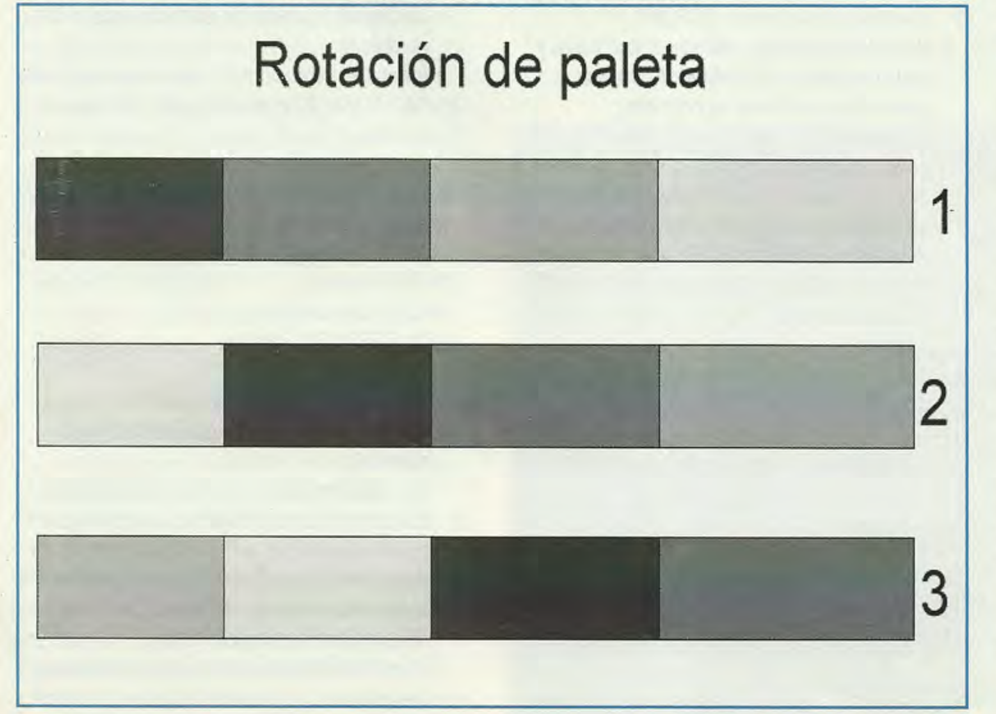
Los objetos *DirectDrawPalette* reservan las entradas 0 y 255 de las paletas de 8 bits, siempre y cuando no especifiquemos el parámetro *DDPCAPS_ALLOW256* que nos permite usar el juego completo de 256 entradas.

Las entradas de paleta se obtienen mediante el método *IDirectDrawPalette::GetEntries*, y se modifican mediante *IDirectDrawPalette::SetEntries*.

PALETAS Y SURFACES NO PRIMARIAS

Las paletas pueden ser usadas en cualquier tipo de *surface* (primaria, *back buffer*, texturas...) pero sólamente las utilizadas en la *surface* primaria tendrán efecto sobre la paleta del sistema de vídeo. Es muy importante notar que las funciones de *blit* de DirectDraw no efectúan transformación de color, lo que implica que las paletas de *surfaces* no primarias sean ignoradas durante este proceso. Las paletas no primarias están pensadas para ser usadas bajo Direct3D.

FIGURA 1. EN ESTA IMAGEN SE MUESTRA EL RESULTADO DE UNA ROTACION DE PALETA.



COMPARTIENDO PALETAS

Las paletas pueden ser compartidas entre múltiples *surfaces*. La misma paleta puede ser vinculada al *front* y al *back buffer* o vinculada a múltiples texturas. Cuando una aplicación vincula una paleta a una *surface* mediante el método *IDirectDrawSurface3::SetPalette*, la *surface* incrementa la cuenta de referencias a esa paleta. Cuando la cuenta de referencias de una *surface* llega a cero, ésta decrementará la cuenta de su paleta vinculada. Asimismo, si una paleta es desvinculada de una *surface* usando *IDirectDrawSurface3::SetPalette* con *Null* como puntero al interfaz *IDirectDrawPalette*, la cuenta de referencias de la paleta de la *surface* se decrementará.

Para realizar la animación de una paleta hay que modificar la paleta para cambiar el modo en que se visualiza la pantalla

Nótese que si llamamos varias veces a *IDirectDrawSurface3::SetPalette* en la misma *surface* y con la misma paleta, la cuenta de referencias de dicha paleta será incrementada una sóla vez. Las siguientes llamadas no afectarán al estado de dicha cuenta.

ANIMACION DE PALETAS

La animación de paleta consiste en modificar la paleta de una *surface* para cambiar el modo en el cual se visualiza en pantalla. Si modificamos repetidamente la paleta, la *surface* parece que cambia 0 sin que efectuemos ninguna modificación en el contenido de la misma. Se pueden simular muchísimos efectos mediante simples animaciones de paleta, tales como fuego, lava y corrientes de agua. Para este fin, este método permite animaciones simples, que no modifican las *surfaces* y no consumen apenas

Principales funciones de regiones de Win32

- **CombineRgn**
  Crea una región resultante de una combinación de dos regiones. Se pueden combinar mediante operadores AND, OR, XOR, DIFF y COPY.
- **CreateRectRgn**
  Crea una región rectangular.
- **CreateRectRgnIndirect**
  Crea una región rectangular a partir de una estructura RECT.
- **GetRegionData**
  Obtiene información de una región.
- **GetRgnBox**
  Obtiene el rectángulo en el que se contiene la región (bounding box).
- **OffsetRgn**
  Desplaza una región.
- **PtInRegion**
  Comprueba si un punto dado está dentro de una región.
- **RectInRegion**
  Comprueba si un rectángulo está todo o en parte dentro de una región.

tiempo de proceso. En el caso más normal, sólo hay que mover 768 bytes (256 colores x 3 bytes RGB).
Existen dos métodos para efectuar animación de paleta:

- Modificando las entradas de una única paleta
- Usando múltiples paletas

En juegos de primera fila como Dune, hay una persona dedicada exclusivamente a diseñar las paletas y sus animaciones

En el primer método hay que ir modificando las entradas que queramos animar de la paleta para posteriormente establecerlas mediante una llamada a *IDirectDrawPalette::SetEntries*. Este es el método que hemos usado en el programa de ejemplo de este mes, aunque nuestra animación se limita a una única entrada de paleta. En el cuadro podemos ver la implementación del efecto.
En el segundo método necesitamos más de un objeto *DirectDrawPalette*. La animación la realizamos vinculando una paleta tras otra a la *surface* mediante llamadas a *IDirectDrawSurface3::SetPalette*.
Ninguno de los dos métodos consume mucho tiempo de proceso, más bien todo lo contrario; por tanto, es decisión vuestra utilizar uno u otro. El primero es más simple y para hacer sencillas rotaciones es menos engorroso. En cambio, para animaciones más complejas, el segundo método es más poderoso. Incluso se

Funciones de la interfaz IDirectDrawPalette por categoría

Funciones de memoria	Initialize
Capacidades de paleta	GetCaps
Entradas de paleta	GetEntries SetEntries

podría programar un editor de animaciones de paleta para situaciones muy complejas. Todo depende del provecho que pensemos sacar del sistema. ¿Alguno recuerda el famoso juego *Dune*, de Cryo? En los créditos aparecía una persona dedicada exclusivamente a diseñar las paletas y sus animaciones. Es un ejemplo inmejorable del buen uso de esta técnica, ya que los efectos que consiguió son buenísimos. Aparte de nuestro programa de ejemplo, en ddex5.exe del SDK de DirectX podemos ver un buen ejemplo de implementación del primer método de animación.

FUNCIONES DE PALETA

- **IDirectDraw2::CreatePalette**

HRESULT CreatePalette(
DWORD dwFlags,
LPPALETTEENTRY lpColorTable,
*LPDIRECTDRAWPALETTE FAR *lplpDDPalette,*
*IUnknown FAR *pUnkOuter*
);

- ***dwFlags***: Tenemos dos *flags* de interés. Si especificamos DDPCAPS_8BIT, crearemos una paleta de 256 colores. Un aspecto a tener en cuenta es el parámetro *DDPCAPS_ALLOW256*, que nos permite utilizar todos y cada uno de los 256 colores de nuestra paleta. En caso de no especificarlo, DirectDraw se reservará las entradas 0 y 255 para uso interno.
  Nota: La función *DDLoadPalette2* de DDUtils crea la paleta sin el parámetro *DDPCAPS_ALLOW256*, con lo cual, si la usamos hemos de tener cuidado ya que tenemos el color 0 y el 255 inservibles.
- ***lpColorTable***: Puntero a un array de 2, 4, 16 o 256 estructuras *PALETTEENTRY* que servirán para iniciar la tabla de colores de la paleta.
- ***lplpDDPalette***: Dirección de un puntero que será rellenado con la dirección del objeto *DirectDrawPalette* creado, en caso de no encontrar problemas al crearlo.
- ***pUnkOuter***: Usado para futura compatibilidad con métodos de agregación de objetos COM. Actualmente, la función fallará si pasamos algo distinto de *Null*..

Funciones de la interfaz IDirectDrawClipper por categoría

Memoria	Initialize
Listas de recorte	GetClipList IsClipListChanged SetClipList
Handles	GetHWnd SetHWnd

Es imprescindible conocer bien todas las funciones de paleta de que disponemos

- **IDirectDrawSurface3::SetPalette**

HRESULT SetPalette(
LPDIRECTDRAWPALETTE lpDDPalette
);

- ***lpDDPalette***: Dirección del objeto *DirectDrawPalette* que será vinculado a la *surface.*

- **IDirectDrawPalette::SetEntries**

HRESULT SetEntries(
DWORD dwFlags,
DWORD dwStartingEntry,
DWORD dwCount,
LPPALETTEENTRY lpEntries
);

- ***dwFlags***: Actualmente no usado. Ha de ser 0.
- ***dwStartingEntry***: Indica la primera entrada de la tabla de colores que será cambiada.
- ***dwCount***: Número de entradas que serán cambiadas.
- ***lpEntries***: Dirección del *array* de estructuras *PALETTEENTRY* que contienen los nuevos colores.

- **IDirectDrawPalette::GetEntries**

HRESULT GetEntries(
DWORD dwFlags,
DWORD dwBase,
DWORD dwNumEntries,
LPPALETTEENTRY lpEntries
);

- ***dwFlags***: No usado. Debe ser 0.
- ***dwBase***: Primera entrada que será obtenida.
- ***dwNumEntries***: Número de entradas que caben en el *array lpEntries*. Los colores son retornados en secuencia, desde el valor de *dwBase* hasta *dwCount*-1 (estos parámetros son establecidos en la llamada a *IDirectDrawPalette::SetEntries*).
- ***lpEntries***: Dirección del *array* de entradas de paleta.



Estructura PALETTEENTRY

```
typedef struct tagPALETTEENTRY { // pe

    BYTE peRed;
    BYTE peGreen;
    BYTE peBlue;
    BYTE peFlags;
} PALETTEENTRY;
```

CLIPPERS COMPLEJOS

En entregas anteriores, vimos cómo crear clippers simples y cómo usarlos. Con ellos podíamos definir una ventana de recorte para los gráficos.
Aprendimos a definirlos usando un identificador de ventana, pero aún nos queda aprender a crearlos en base a una lista de rectángulos. Ello nos permitirá crear efectos como cortinillas, optimizar dibujados y muchas cosas más que se nos vayan ocurriendo. Las posibilidades son muchas.

Para crear clippers complejos tendremos que usar el método IDirectDrawClipper y las funciones de regiones

Por ejemplo, en nuestro programa hemos usado un clipper complejo para presentar un texto móvil en dos esquinas de la pantalla. Sin el uso de este método, el efecto hubiese sido muchísimo más engorroso de realizar, ya que hubiésemos tenido que utilizar una *surface* auxiliar para efectuar el recorte, con el consiguiente tiempo de proceso perdido en los dibujados extra.
Para crear los *clippers* complejos hemos de usar dos elementos: el método *IDirectDrawClipper::SetClipList* y las funciones de regiones del API Win32.

- **IDirectDrawClipper::SetClipList**

```
HRESULT SetClipList(
LPRGNDATA lpClipList,
DWORD dwFlags
);
```

  - ***lpClipList***: Dirección a una estructura *RGNDATA* válida o *Null*. Si existe una lista de *clipping* en el objeto y este parámetro vale NULL, dicha lista será borrada.
  - ***dwFlags***: Actualmente no usado. Debe ser 0.

De las funciones de regiones, cabe destacar dos de ellas que nos permitirán crear todo tipo de listas de *clipping* basadas en rectángulos. Cabe señalar que, aunque el API Win32 permite la creación y manejo de regiones con elipses y curvas, los *clippers* de DirectDraw sólo admiten regiones basadas en rectángulos. Las funciones que usaremos son *CreateRectRgn* y *CombineRgn*.

- **CreateRectRgn**
Esta función crea una región basada en un único rectángulo.

```
HRGN CreateRectRgn(
  int nLeftRect,
  int nTopRect,
  int nRightRect,
  int nBottomRect
);
```

  - ***nLeftRect***: Especifica la coordenada horizontal de la esquina superior izquierda.
  - ***nTopRect***: Especifica la coordenada vertical de la esquina superior izquierda.
  - ***nRightRect***: Especifica la coordenada horizontal de la esquina inferior derecha.
  - ***nBottomRect***: Especifica la coordenada vertical de la esquina inferior derecha.

- **CombineRgn**
Esta función combina dos regiones y guarda el resultado en una tercera región. El modo de combinación es especificado en el último parámetro.

```
int CombineRgn(
  HRGN hrgnDest,
  HRGN hrgnSrc1,
  HRGN hrgnSrc2,
  int fnCombineMode
 );
```

  - ***hrgnDest***: Identifica a una región que será la combinación de las otras dos.

Ejemplo de animación de una entrada de paleta

Animamos una entrada de paleta. El objeto lpddPaleta es la paleta principal.

```
Static int scroll = 0;
Static int scrollDir = 1;

Static int col = 0;
Static int colDir = 4;

PALETTEENTRY        pal;

//
//          Animamos un color de la paleta...
//
col+=colDir;
if(col>255)
{
            col=255;
            colDir*=-1;
}
if(col<0)
{
            col=0;
            colDir*=-1;
}

pal.peRed = (BYTE)col;
pal.peBlue = 0;
pal.peGreen = 0;
pal.peFlags = 0;

lpDDPaleta->SetEntries(0, 253, 1, &pal);
```

Ejemplo de carga y uso de paletas

Objetos necesarios: una paleta y una surface

```
LPDIRECTDRAWPALETTE lpDDPaleta;
    // Paleta del ejemplo
LPDIRECTDRAWSURFACE3          lpDDS;
```

Cargamos la paleta mediante DDLoadPalette2

```
// Cargamos la paleta desde un recurso BMP
if((lpDDPaleta =
DDLoadPalette2(directDraw.get(), "PALETA"))==
NULL)
{
    PRINTD("FALLO AL CARGAR LA PALETA")
}
```

Vinculamos la paleta a una surface

```
Else
{

    // Si se ha cargado, la establecemos en la
surface
    if(FAILED(lpDDS->SetPalette(lpDDPaleta)))
    {
            PRINTD("FALLO AL ESTABLECER LA
PALETA")
    }

}
```

Tras usar la paleta, la liberamos

```
// Libera la paleta
if(lpDDPaleta!=NULL)
lpDDPaleta->Release();
```

Ejemplo de creación de clippers simples y complejos

```
Void creaClipper(void)
{

    //
    //          Declaramos los objetos
necesarios...
    //
    LPDIRECTDRAW2     lpDD;
    LPDIRECTDRAWSURFACE3      lpDDS;
    HRGN region, reg1, reg2;
    LPRGNDATA rgnData;


    lpDD = directDraw.get();
    lpDDS = directDraw.getBack()->get();

    // Crea los clippers del programa
    lpDD->CreateClipper(0,&lpDDClip,NULL);
    lpDD->CreateClipper(0,&lpDDClipAll,NULL);

    // Creamos una region con dos rectángulos
    region = CreateRectRgn(0,0,1,1);          //
Destino: debe ser creada
    reg1 = CreateRectRgn(0,0,199,99);         //
Rectangulo 1
    reg2 = CreateRectRgn(440,380,639,479);
// Rectangulo 2
    CombineRgn(region, reg1, reg2, RGN_OR);

    // Las combinamos

        if(region)
        {
                //
                // Si ha ido todo bien, le pasamos
la región al clipper...
                //
                rgnData = (LPRGNDATA)new
char[GetRegionData(region, 0, NULL)];
                GetRegionData(region,
GetRegionData(region, 0, NULL), rgnData);

                // Le pasamos la lista de
rectángulos...
                lpDDClip->SetClipList(rgnData, 0);
                delete rgnData;
        }
    // Si algo ha ido mal, creamos un clipper de
ventana completa...
    else lpDDClip->SetHWnd(0,
directDraw.getHWND());

    // Nuestro segundo clipper es de ventana
completa...

    lpDDClipAll->SetHWnd(0,
directDraw.getHWND());
```

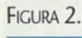


FIGURA 2.



FIGURA 3.

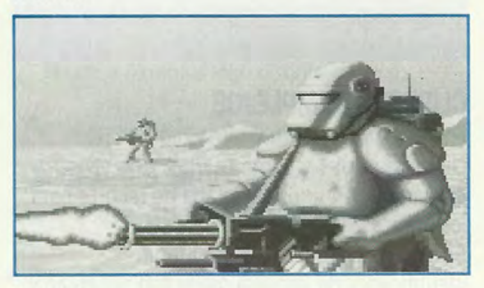


FIGURA 4.

Es importante advertir que es necesario que esta región exista de antemano.

- *hrgnSrc1*: La primera de las dos regiones que será combinada.
- *hrgnSrc2*: La segunda de las regiones que será combinada.
- *fnCombineMode*: Especifica la manera de la cual se combinarán las dos regiones. Podemos usar cualquiera de estos parámetros:
  - RGN_AND: Crea la intersección de las dos regiones.
  - RGN_COPY: Crea una copia de la región 1.
  - RGN_DIFF: Combina las partes de *hrgnSrc1* que no pertenecen a *hrgnSrc2*.
  - RGN_OR: Crea la unión de las dos regiones.
  - RGN_XOR: Crea la unión de las regiones excepto en las zonas donde coincidan.

  Nota: Las tres regiones no tiene por qué ser distintas. Por ejemplo, *hrgnSrc1* puede ser igual que *hrgnDest*.

De las funciones de regiones destacan dos que nos permitirán crear listas de clipping basadas en rectángulos

EL PROGRAMA DE EJEMPLO

En el programa de este mes vemos ejemplos de animación de paleta y uso de *clippers* complejos. La animación es algo simple, ya que sólo se anima una entrada (la 253), aunque resultaría fácil usar algunas más.

Y ya sabéis: para dudas, sugerencias o cualquier tipo de críticas, el que suscribe estas líneas estará encantado de responderos. Nos veremos el mes que viene, en esta misma sección.

Autor: Carlos Glez. Morcillo

Wolfenstein 3D

Segunda entrega en esta sección. Tenemos nada menos que el código fuente completo del juego que supuso una auténtica revolución en videojuegos 3D hace 6 años. Naturalmente estamos hablando del Wolfenstein 3D (de Id Software).

Para los que se hayan incorporado hace poco al mundo del videojuego en PC, podemos decirle que Wolfenstein fue un juego que marcó el comienzo de una era. Allá por 1992, cuando los ordenadores domésticos podían llegar a ser 286 apareció esa joya. Parecía imposible que en un 80286 se pudiera disfrutar de un juego 3D con texturas y que fluyera suavemente en la pantalla. ¿Era cosa de magia?, si fuera así, John Carmack y John Romero (los programadores) eran brujos profesionales. Después del Wolfenstein la lluvia de juegos basados en esta técnica ha crecido casi exponencialmente. La saga del Doom (de los mismos programadores), Quake y una larga lista de clones han ido apareciendo en el mercado con mayor o menor fortuna. Lo que no podemos negar es que el Wolf cambió el concepto de videojuego de PC.

El juego fue desarrollado en C++ (utilizando el compilador de Borland 3.0) y en Ensamblador. John Carmack utilizó una sencilla pero efectiva técnica de Ray Casting para el entorno 3D (él mismo admite que se podrían mejorar muchas cosas, pero el juego tenía que correr en un 826). El modo de vídeo es el archiconocido 13h (tenían que soportarlo las tarjetas VGA de la época), y se han dejado algunas opciones que podremos añadir al programa como:

- Añadir nuevos enemigos y armas.
- Convertir el código para un compilador de 32 bit.
- Opciones de multi-jugador (podremos usar las mismas rutinas que el Doom).
- Soporte para movimientos verticales (saltar y subirse a objetos).
- Posibilidad de hacer habitaciones mucho más altas.

El primer pilar de uno de los géneros más populares de todos los tiempos se nos presenta sorprendente, inteligente y sencillo

Para añadir las dos últimas opciones haría falta reescribir la parte del código referente al mapeado de texturas.

Para compilarlo os harán falta ficheros de gráficos, música... que podréis obtener en la versión completa de Wolfenstein 3D o, si no disponeis de ella, también podréis compilarlo con los ficheros de la versión demo que encontraréis en el CD de la revista.

Aunque es un juego bastante antiguo, siempre podremos aprender bastantes cosas sobre sus fuentes. Un documento sin desperdicio de dos gurús de la programación de videojuegos (ejemplo seguido por muchos programadores). Y con esto y un bizcocho... ¡¡¡hasta el mes que viene!!!

No queremos irnos sin añadir que sentimos no haber incluído el informe de Anarkano en el interior del CD-Rom del mes pasado. El error ha sido convenientemente subsanado en el de este mes. Esperamos que podáis perdonarnos un error semejante. Estamos seguros de que sí.

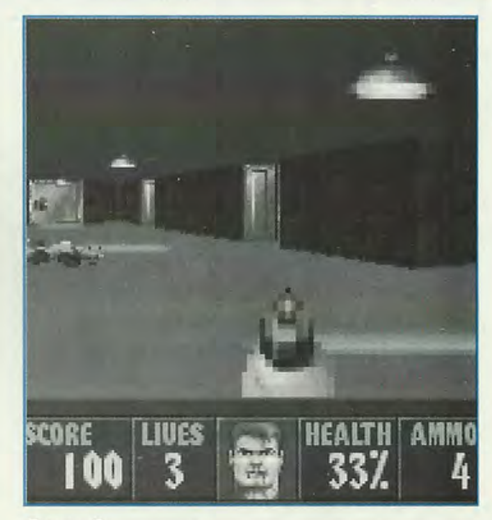


FIGURA 5.

FIGURA 1.



FIGURA 2.

FIGURA 3.



FIGURA 4.



Autor: Carlos Gzlez. Morcillo

Curso de animación (I)

Una buena animación de personajes es fundamental para el éxito de un videojuego. El dotar de vida propia a un conjunto de pixel es una labor que requiere unos conocimientos teóricos y bastantes horas de práctica. En el curso que comenzamos este mes os guiaremos por los senderos teóricos de la animación, las horas de práctica correrán de vuestra cuenta.

En esta primera entrega, antes de meternos de lleno a dibujar, expondremos un poco el objetivo de este curso, las partes de las que va a contar y el material que nos hará falta.
La animación por ordenador está cobrando cada día más importancia. Tanto si utilizamos el ordenador como si no, constantemente recibimos gran cantidad de información visual generada por ordenador; tanto la televisión, el cine, publicidad, páginas web y, cómo no, los videojuegos contienen gran cantidad de animación por ordenador. La demanda en el mercado laboral de profesionales que dominen la animación por ordenador está creciendo cada día. Este curso no pretende (ni podría pretender en tan poco espacio) ser una guía para convertirse en un animador profesional, pero sí servirá para que los interesados en este campo adquieran un buen nivel de conocimientos.
El curso tendrá 8 entregas, en las primeras aprenderemos a crear personajes a partir de formas básicas, dominar las expresiones faciales (para dotarle de distintos estados de ánimo) y los movimientos de la boca cuando hablan. En general en las tres primeras entregas aprenderemos a crear personajes, repasaremos las principales técnicas de animación y animaremos el rostro de los mismos (en diálogos y primeros planos). Una vez explicado esto, pasaremos a la animación de cuerpo entero, distintos movimientos, líneas de acción... Esta segunda parte nos llevará otras tres entregas más. En la última parte del curso nos centraremos en la perspectiva, integración de los personajes en un fondo, acciones simultáneas de varios personajes e introduciremos algunos conceptos avanzados.

La animación en los videojuegos, uno de los mundos más apasionantes en este campo, es analizada teóricamente

Los ejemplos del curso los iremos aplicando de diferentes formas, así, dominaremos más campos de la animación. Algunos los animaremos en 2D como si fueran destinados a un videojuego de plataformas o similar. A otros ejemplos les daremos otro tratamiento para destinarlos a formar parte de una página web. También tocaremos la animación 3D, para juegos tipo Quake, o de estrategia en 3D. Estos últimos ejemplos los veríamos en la sección Taller 3D.

FIGURA 1.

HERRAMIENTAS DE DESARROLLLO

Las herramientas que utilizaremos para animar serán básicamente tres. La primera, y en la que más ejemplos desarrollaremos, será el *Autodesk Animator Pro*. Esta aplicación para MS-DOS tiene bastantes años, pero sin embargo creemos que no ha aparecido actualmente ninguna que la supere en PC para realizar animaciones de videojuegos. El Autodesk Animator Pro tiene la posibilidad de insertarle scripts que el programador de nuestro grupo de desarrollo confeccione en C para trabajar más cómodamente, manejar tipos de ficheros propios... (posiblemente expliquemos en una entrega cómo se realizan estos scripts). El único inconveniente que le vemos a este programa es que no permite trabajar con más de 256 colores, pero, para animar un personaje, (salvo contadas excepciones) nos sobrarán con 8 bits de color. Además de tener un interfaz de usuario muy amigable, ofrece posibilidades de animación muy interesantes. Sin duda, la mejor herramienta para animar personajes 2D de videojuegos.
También utilizaremos el Adobe Photoshop 5.0 para retocar algunas imágenes y realizar paisajes, elementos estáticos y algunos efectos puntuales. Dependiendo de vuestra respuesta le dedicaremos más o menos tiempo (¿qué tal un curso independiente de Photoshop?).
Nos valdremos del 3D Studio Max 2.5 para animar personajes en 3D. Posiblemente utilicemos algunos plug-ins externos para



FIGURA 2.

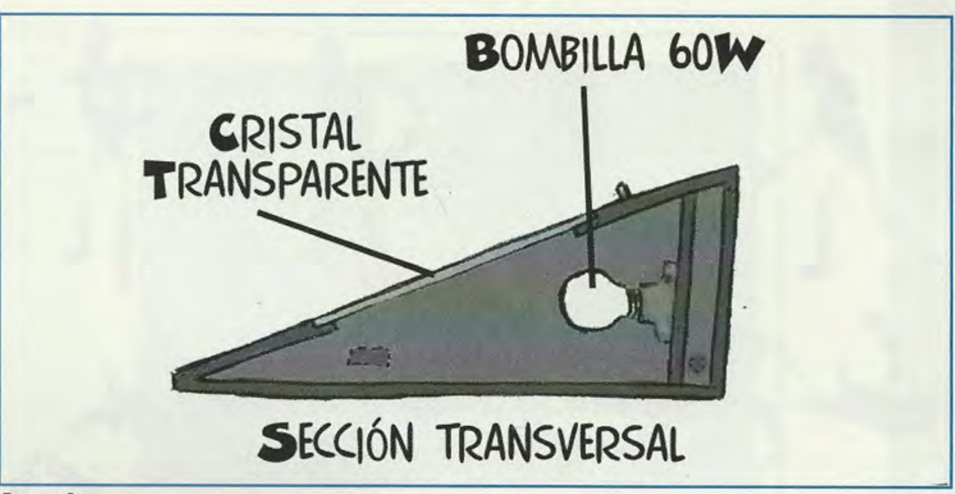


FIGURA 3.

facilitarnos el trabajo (de los cuales podréis encontrar una demo o versión completa en el CD de la revista del número correspondiente).

La elección entre animación 2D y 3D dependerá del proyecto y el tiempo que tengamos

Para seguir el curso necesitaremos material "hardware". Cualquier animador 2D debería tener una tabla de animación (también denominada "mesa de luz") para trabajar más rápidamente. En el dibujo tenéis un esquema para construir vuestra propia mesa de luz. En la parte posterior de la mesa, sería aconsejable que realizarais unas ranuras para que pueda entrar un poco el aire porque, cuando se lleva un rato dibujando, la mesa se calienta un poco. Como animar directamente con el ratón en la pantalla, puede ser un trabajo de chinos, nosotros seguiremos otra forma de trabajar. Primero dibujaremos al personaje en papel en nuestra mesa de luz, después escanearemos los diferentes *frames* y finalmente los retocaremos y colorearemos en el ordenador. Todo esto lo explicamos con más detenimiento en la siguiente lección del curso.
Respecto al papel que utilizamos para dibujar, tendrá que ser blanco y no demasiado grueso (el de baja calidad nos vendrá bien, de 60 gramos o incluso menos). Lo ideal sería disponer del celuloide que utilizan en los estudios de animación, pero se nos podría disparar el presupuesto. El tamaño máximo que utilizamos será A4, aunque posiblemente nos resulte más cómodo trabajar en A5. Preparamos el papel con dos agujeros en la parte superior que deben encajar con los dos palitos que sobresalen de la mesa de luz. Además de la tabla de animación, sería bueno que dispusiéramos de un espejo de nuestro tamaño. Así, podremos realizar el movimiento que queremos dar al personaje nosotros mismos y podremos observar nuestros movimientos en el espejo. Es la mejor forma de hacernos una idea de los gestos que tendrá que hacer nuestro personaje en determinadas acciones. De cualquier forma, lo ideal es que alguien posara y realizara esos movimientos para nosotros (¿conseguiréis engañar a alguien?) Podremos ayudarnos de una cámara de vídeo y grabarnos realizando cada movimiento. Después, en nuestro lugar de trabajo, los podemos ver en una televisión una y otra vez hasta que captemos todos los movimientos a realizar. En los estudios de Walt Disney trabajan de una manera similar. Por citar un ejemplo, los enanitos de la película Blancanieves fueron animados observando los movimientos de personas que posaron y fueron grabadas para ese fin. En los famosos estudios de animación Pixar, para hacer Toy Story también contaron con modelos reales.

ANIMACION 2D Y ANIMACION 3D

La utilización de un tipo de animación u otra dependerá del proyecto que estemos realizando y del tiempo del que dispongamos

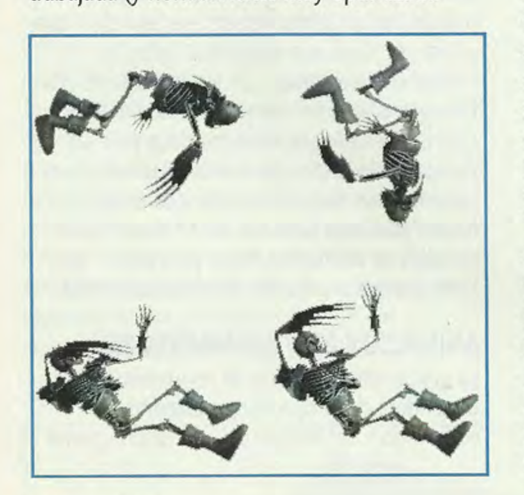
FIGURA 4.





FIGURA 5.

para el desarrollo. Aunque a priori parezca más costosa de realizar una buena animación 2D, realmente requiere menos tiempo que una 3D. Las razones son obvias: el personaje 3D necesita una malla que deberá estar muy trabajada (y normalmente muy optimizada para que el número de polígonos que tenga pueda ser manejado sin problemas por el motor del juego), además de una buena textura (aspecto crítico en el que debemos invertir bastante tiempo). El personaje 2D, sin embargo, se dibuja y listo. Para videojuegos donde los movimientos de los personajes no sean muy variados podríamos optar por animar en 2D, mientras que si el ángulo de visión cambia o los movimientos son muy variados debemos animar en 3D. Una vez construido el personaje 3D podremos utilizarlo para animarlo y ponerlo en miles de posturas distintas en "poco" tiempo, mientras que si lo dibujamos en 2D, cada modificación que hagamos en él supondrá un nuevo dibujo. Por lo tanto, si el grafista puede elegir entre diseñar 2D ó 3D (es decir, el proyecto no impone un tipo de diseño), elegiremos 2D cuando no haya demasiados movimientos distintos o cuando el tiempo se nos venga encima y 3D cuando queramos reutilizar el modelo más veces.



FIGURAS 6 Y 7.



TERMINANDO POR ESTE MES...

Aquí concluimos esta pequeña introducción al curso de animación. En el temario se nos quedarán bastantes cosas en el tintero por razones de espacio en la revista. Si de algunos temas tratados queréis más información, podéis contactar con nosotros y hacernos llegar vuestra opinión. Recordar que para seguir el curso necesitaréis una tabla de animación (si la construís vosotros mismos no os costarán más de 2000 pesetas los componentes) y opcionalmente (pero muy aconsejable) el espejo o la cámara de vídeo. Utilizaremos Autodesk Animator Pro, Adobe Photoshop 5 y MAX 2.5, aunque intentaremos incluir en el CD de la revista programas shareware y freeware con los que realizaremos algunos ejemplos de los artículos. Empezar a afilar vuestros lapiceros, sacar brillo a vuestros ratones (o tablas digitalizadoras) que el mes que viene comenzamos de lleno. Hasta entonces, un saludo desde la redacción.

El personaje de una animación 3D requiere una malla que tendrá que estar perfectamente elaborada

